NOTICE OF CHANGE

NOT MEASUREMENT SENSITIVE

MIL-STD-2407
NOTICE 1
26 October 1999

DEPARTMENT OF DEFENSE INTERFACE STANDARD

VECTOR PRODUCT FORMAT

TO ALL HOLDERS OF MIL-STD-2407:

1. THE FOLLOWING PAGES OF MIL-STD-2407 HAVE BEEN REVISED AND SUPERSEDE THE PAGES LISTED:

NEW PAGE	DATE	SUPERSEDED PAGE	DATE
23	26 October 1999	23	28 June 1996
24	26 October 1999	24	28 June 1996
43	26 October 1999	43	28 June 1996
44	28 June 1996	44	Reprinted without change
45	28 June 1996	45	Reprinted without change
46	26 October 1999	46	28 June 1996
53	28 June 1996	53	Reprinted without change
54	26 October 1999	54	28 June 1996
79	26 October 1999	79	28 June 1996
80	28 June 1996	80	Reprinted without change
121	26 October 1999	121	28 June 1996
122	26 October 1999	122	28 June 1996
123	26 October 1999	123	28 June 1996
124	28 June 1996	124	Reprinted without change
159-172	26 October 1999	159-172	28 June 1996 Appendix F

2. RETAIN THIS NOTICE AND INSERT BEFORE TABLE OF CONTENTS.

3. Holders of MIL-STD-2407 will verify that page changes and additions indicated above have been entered. This notice page will be retained as a check sheet. This issuance, together with appended pages, is a separate publication. Each notice is to be retained by stocking points until the standard is completely revised or canceled.

Custodians:
Army – TI
Navy – NO
Air Force – 09

Review activity:
Marine Corps - MC

Preparing activity:
NIMA – MP
(Project MCGT-0338)

row ids shall start at 1 and be sequential with no gaps in the numbering. TABLE 2 depicts the principal components of a VPF table.

TABLE 2. VPF table structure.

Table Header	
Metadata and column definitions: a. Table description b. Narrative table name (optional) c. Column definitions: Column name Field type Field length Key type Column textual description Optional value description table name Optional thematic index name Optional column narrative table name	
id	**table contents**
Indicates the starting position of each row.	The data composing the table that match the column definitions.

This document describes the column definitions for all the VPF standard–specified columns, and the table organization for those columns. No specific ordering of columns within a table is required. Product specifications may require a particular product-specific order. Data columns and tables described in this document are labeled either mandatory or optional. A VPF product must include all mandatory tables and columns. It is not possible to remove any mandatory column from any table. A VPF-compliant application must be able to process a VPF product and interpret all mandatory and optional columns as described in this document.

Additional product-specific columns are allowed by VPF. If present, these columns must be defined in their product specifications. Product-specific columns must not alter the use of the columns specified in this document.

5.2.1.4 Indexes. A table may have associated indexes. If a table contains a variable-length coordinate string column, a variable-length string column, or a field type “K” (triplet id), a separate variable-length index file must be present.

In addition to variable-length indexes, VPF also supports spatial, thematic, and feature indexes. Spatial indexes contain references to row data that are based on the value of a coordinate column. Thematic indexes contain references to row data that are based on the value of non-coordinate columns. Feature indexes have been developed to enhance processing of complex queries. They contain references linking rows in primitive tables to rows in associated feature tables.

5.2.1.5 Narrative tables Each VPF table may have an associated narrative table that provides miscellaneous information about the VPF table. The purpose of the narrative table is to provide the database designer with the ability to record comments or information pertinent to the associated table. The narrative table name is stored in the VPF table's header information. In addition, VPF provides for optional narrative tables keyed to individual columns within a table. The narrative table name is stored as the third optional entry in the column definition (see TABLE 2).

5.2.1.6 Attribute tables. Real-world objects are referred to as entities or features; they are modeled in tables in VPF. The properties of entities are called attributes. In an attribute table, one table column is defined for each attribute describing an object. Each object occupies a row in the table. Examples of attributes include data quality, size, and name. A sample attribute table is shown in TABLE 3.

A column or a group of columns that can be used to identify or select a row is called a key. A unique key is a key that uniquely identifies each row. One unique key is designated the primary key; each table has one and only one primary key. In the city attribute table (TABLE 3), the built-up area column is the primary key.

TABLE 3. City attribute table.

id	**built-up area**	**state**	**population size**	**median income per household**
implicit	*character string*	*character string*	*binary integer*	*binary integer*
UNIQUE KEY	PRIMARY KEY	NON-UNIQUE	NON-UNIQUE	NON-UNIQUE
1	Los Angeles	California	2966850	15735
2	New York	New York	7071639	13854
3	Salt Lake City	Utah	163033	13211
4	Las Vegas	Nevada	164674	17468
5	San Francisco	California	1366383	16782

a. Edges: When an edge is broken by a tile boundary a connected node is placed at the edge-tile intersection. The identical (in terms of a geographical coordinate tuple) connected node occurs in all adjacent tiles forming the boundary. All edges which lie along a tile boundary, will have cross-tile topology. The identical (in terms of a geographical coordinate tuple) edge occurs in both tiles forming the boundary.

b. Faces: A face broken by a tile boundary has a new edge constructed and inserted at the boundary for each tile to close the face internal to the tile. These edges take part in cross-tile topology.

c. Face 1: Face 1 (universe face) represents a special case for tile boundaries. In those cases where face 1 is the only face being broken, actual tile boundaries will not be stored. For example, where face 2 is broken by the tile boundary and the rest of the tile is defined by face 1, only the tile boundary edges necessary to close face 2 are stored (FIGURE 13).

d. Connected Nodes: All connected nodes which lie on a tile boundary will have cross-tile components (tile_id and first_edge).

Two other situations to consider are that of a tile of a level 3 topology coverage which contains only point features or no features. In these cases, the tile contains either entity node primitives and face 1 or simply face 1, respectively. Level 3 topology requires inclusion of a face, ring, edge, connected node and face bounding rectangle table, and an edge variable-length index (TABLE 8). The face, ring and face bounding rectangle tables will reference the universe face (face 1) only. The edge table must exist since it is referenced by the ring table. The connected node table must exist since it is referenced by the edge table. The existence of an edge table requires an edge variable-length index and the existence of a face table requires a face bounding rectangle table. The edge and connected node table and the edge variable-length index will contain no records. See TABLE below for this scenario.

fac table		
id	dnarea.aft_id	ring_ptr
1	(-2147483648)	1

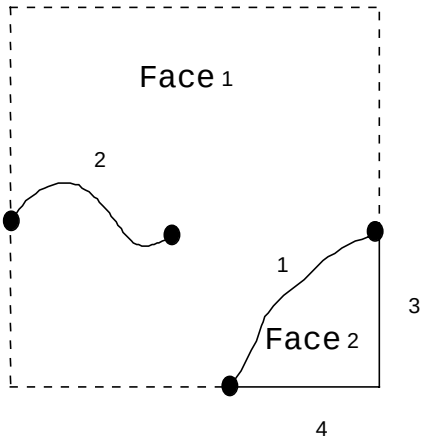
rng table		
id	fac_id	start_edge
1	1	(-2147483648)

fbr table				
id	x min	y min	x max	y max
1	(null)	(null)	(null)	(null)

edg table						
id	dnline.lft_id	sn	en	rf	lf	coordinates
(no records---header information only)						

cnd table				
id	dnpoint.pft_id	containing_face	first_edge	coordinates
(no records---header information only)				

end table			
id	dnpoint.pft_id	containing_face	coordinates
1	1	1	37.5, -76.5
2	5	1	39.0, -80.0



Note: Face 1 is the universe face. The tile's edge file will only store edges 1,2,3 and 4. The dashed edges for the universe face are implied, but not stored.

FIGURE 13. Storage of tile boundaries.

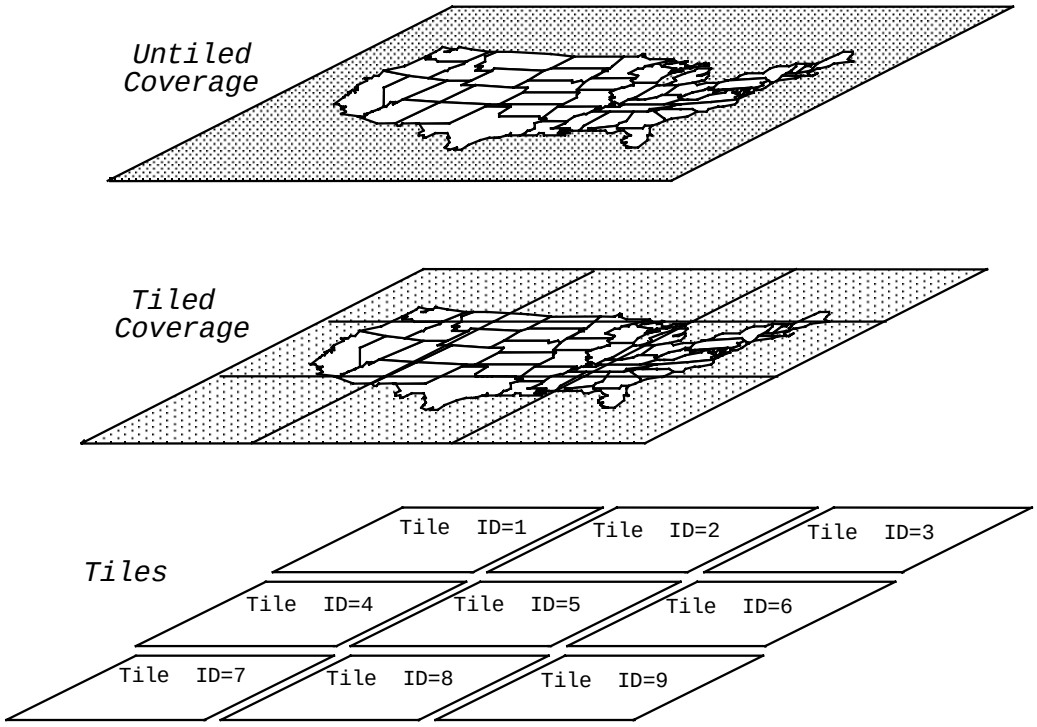


FIGURE 14. A tiling scheme.

5.2.2.3.4 Cross-tile keys. VPF provides a mechanism for maintaining geographic features in a logically continuous spatial database, whether or not a tiling scheme is present. Since the primitives in each tile of a tiled coverage are managed separately from those in other tiles, labels given to primitives are unique only within a tile. In order to support a logically continuous spatial database, a triplet id can be used instead of an integer key to reference primitives across multiple tiles. The triplet id augments the key of a primitive with the key of the tile in which the primitive falls. APPENDIX B contains a discussion that fully describes this concept.

a. For an edge primitive, the triplet id is used to maintain cross-tile topology. The Left Face, Right Face, Left Edge, and Right Edge columns are defined as triplet ids to support tiled coverages. The triplet id contains a reference to the internal topology within the current tile; the two other components reference the external tile directory and the primitive within that tile. For example, for a face divided by a tile boundary, the external id portion of the Left Face field in FIGURE 15 would include the continuing face in the other tile. This inclusion of internal and external tile references allows software to detect tile borders and continue operations across boundaries

or to operate only within the current tile. If a coverage is untiled, the left face, right face, left edge, and right edge columns may be defined as integer columns; otherwise the external tile id and primitive id sub-fields of the triplet id will not exist (see 5.4.6).

b. For a connected node in a tiled coverage the triplet id is applied to the first_edge column.

c. Cross-tile topology only occurs between tiles within a library. Cross-tile components will only be populated for edges intersecting tile boundaries within a library. There is no cross-tile topology between tiles in different libraries.

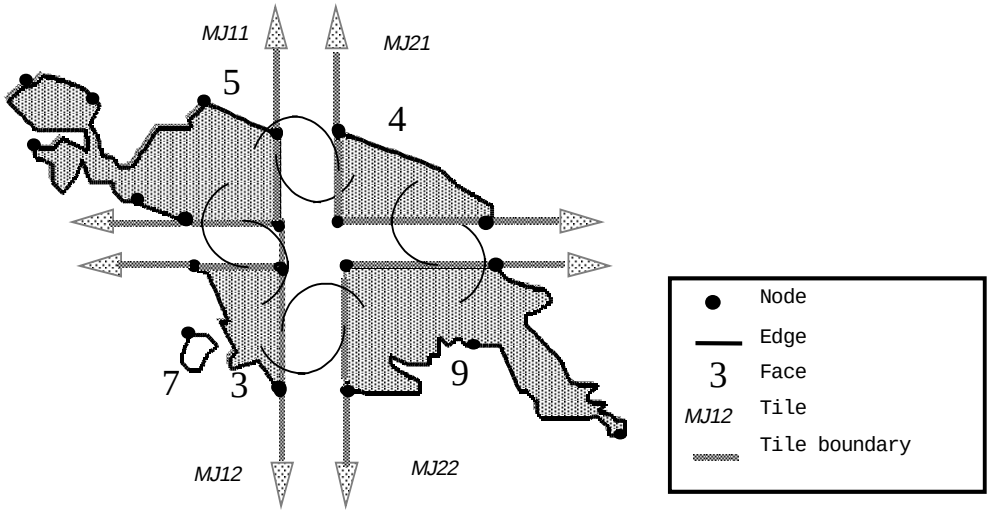


FIGURE 15. Face cross-tile matching.

5.2.2.4 Library. A library is a collection of coverages that share a single coordinate system and scale, have a common thematic definition, and are contained within a specified spatial extent. If any of the coverages composing the library are tiled, then all other coverages must either use the same tiling scheme, or be untiled. The contents and organization of the libraries are determined by a product specification. All of the tables and coverages making up the library are contained within a single master directory (FIGURE 16).

5.3.1.2 Reserved table names and extensions. Each VPF table name consists of a reserved name or suffix extension. TABLE 13 lists the tables whose names cannot be modified or changed.

There are a few reserved directory names at the library and database levels. These names are listed in TABLE 14.

In a coverage directory, there are many feature class tables that have reserved suffixes. The product specification may define any eight-character prefix, following the naming conventions detailed in section 5.4.5. TABLE 15 lists the table suffixes.

TABLE 13. Reserved file names.

File Name	Description
cat	Coverage Attribute Table
cnd	Connected Node Primitive
csi	Connected Node Spatial Index
dht	Database Header Table
dqt	Data Quality Table
ebr	Edge Bounding Rectangle
edg	Edge Primitive
end	Entity Node Primitive
esi	Edge Spatial Index
fac	Face Primitive
fbr	Face Bounding Rectangle
fca	Feature Class Attribute Table
fcs	Feature Class Schema Table
fsi	Face Spatial Index
grt	Geographic Reference Table
lat	Library Attribute Table
lht	Library Header Table
nsi	Entity Node Spatial Index
rng	Ring Table
txt	Text Primitive
tsi	Text Spatial Index
char.vdt	Character Value Description Table
int.vdt	Integer Value Description Table

TABLE 14. Reserved directory names.

Directory Name	Description
libref	Library reference coverage
dq	Data quality coverage
tileref	Tile reference coverage
gazette	Names reference coverage

TABLE 15. Reserved table name extensions.

File Name Suffix	Description
.abr	Area Bounding Rectangle Table
.aft	Area Feature Table
.ajt	Area Join Table
.ati	Area Thematic Index
.cbr	Complex Bounding Rectangle Table
.cft	Complex Feature Table
.cjt	Complex Join Table
.cti	Complex Thematic Index
.doc	Narrative Table
.dpt	Diagnostic Point Table
.fit	Feature Index Table
.fti	Feature Index Table Thematic Index
.jti	Join Thematic Index
.lbr	Line Bounding Rectangle Table
.lft	Line Feature Table
.ljt	Line Join Table
.lti	Line Thematic Index
.pbr	Point Bounding Rectangle Table
.pft	Point Feature Table
.pjt	Point Join Table
.pti	Point Thematic Index
.rat	Related Attribute Table
.rpt	Registration Point Table
.tbr	Text Bounding Rectangle Table
.tft	Text Feature Table
.tjt	Text Feature Join Table
.tti	Text Thematic Index

Any table that contains variable-length records must have a variable-length index associated with it. The index file shall have the same file name as the table, except that the last character will end with "x". For example, a variable-length record road line table, road.lft, would have a variable-length index road.lfx. The one exception to this convention is for the fcs, whose variable-length index shall be named fcz.

5.3.6.2 Database header table. The database header table (TABLE 46) contains information that defines database content and security information.

TABLE 46. Database header table definition.

Column Name	Description of Contents	Column Type	Key Type	Op/Man
id	Row id	I	P	M
vpf_version	VPF version number	T,10	N	M
database_name	Directory name of the database	T,8	N	M
database_desc	Text description of the database	T,100	N	M
media_standard	Media standard used for the database	T,20	N	M
originator	Text for title and address of originator (a backslash "\" is used as a line separator)	T,*	N	M
addressee	Text for title and address of addressee (a backslash "\" is used as a line separator)	T,*	N	M
media_volumes	Number of media volumes comprising the database	T,*	N	M
seq_numbers	Sequential number(s) for each media volume in this database	T,*	N	M
num_data_sets	Number of libraries within database	T,*	N	M
security_class	Security classification of database (the highest security classification of the transmittal including all datasets within the database) T = TOP SECRET S = SECRET C = CONFIDENTIAL R = RESTRICTED (or alternatively "FOR OFFICIAL USE ONLY") U = UNCLASSIFIED	T,1	N	M
downgrading	Originator's permission for downgrading required (yes or no)	T,3	N	M
downgrade_date	Date of downgrading	D	N	M
releasability	Releasability restrictions	T,20	N	M
other_std_name	Free text, note of other standards compatible with this database	T,50	N	O
other_std_date	Publication date of other standard	D	N	O
other_std_ver	Other standard amendment number	T,10	N	O
transmittal_id	Unique id for this database	T,*	N	M
edition_number	Edition number for this database	T,10	N	M
edition_date	Creation date of this database	D	N	M

5.3.7 Data quality. The data quality table may be stored at the database, library, or coverage level. It contains information on the completeness, consistency, date status, attribute accuracy, positional accuracy, and other miscellaneous quality information. It also contains information about the source from which the geographic data was derived. Lineage information should be included in the associated narrative table, named "LINEAGE.DOC." While the contents and location of a data quality table within a VPF database are product specific, it is highly recommended that at least one table exist at the library level. TABLE 47 defines the contents of the data quality table. APPENDIX E contains more information about storing data quality information in VPF.

below illustrates a depth first search. The algorithm is as follows:

a. Locate current edge with user application (edge 12).

b. Read end node of current edge and gather all edges incident at the node.

c. For each of the edges incident at the node, read the attribute value from the associated line feature. Does the attribute have the desired value? If so, continue with step e.

d. Go to step c. Repeat with next edge.

e. Decision. Has the network been completely traversed? If so, exit with complete network. If not, go to step b.

B.4.4 Cross-tile topology. Network navigation using the winged-edge topology can be extended to cross over physical tile partitions, if they exist in the coverage. By using the information in the previous examples, it becomes possible to introduce cross-tile constructs. Assume that FIGURE 32 has been intersected with tile boundaries and that the new coverage in FIGURE 35 has been created (with generalized edges along edge 5 in FIGURE 32).

FIGURE 35 depicts a single face broken into four faces by the intersection of four tiles. The following discussion identifies several different occurrences at the tile boundaries and covers retrieval of the original (untiled) face extent from the tiled faces through winged-edge topology.

When creating cross-tile topology, the following rules apply:

a. An edge is always broken when it intersects a tile boundary by placing a connected node at the intersection in all adjacent tiles. See FIGURE 34, B,C,G and H. All edges terminated by this connected node will have cross-tile topology if an edge exists in the adjoining tile. See FIGURE 33 A through F.

b. The cross-tile edge will be the first edge in the adjacent tile, counterclockwise from the referencing edge at the node. See FIGURE 33, A through F.

c. All edges which are coincident with a tile boundary (all coordinates for the edge are on the boundary) occur in both tiles (see FIGURE 34, A, D, E and F) and have cross-tile left or right face topology and have cross-tile left and right edge topology. See FIGURE 33, A,B, D through F.

d. When a face is broken by a tile boundary, multiple faces are created by closing the face along the tile boundary. See FIGURE 34, A. The edges used to close faces on the boundary are treated as in c. above. See FIGURE 33, B and E.

e. Connected nodes which occur on tile boundaries, exist in all adjacent tiles (see FIGURE 34, B, C, G and H) and reference both an internal and external first edge, if an edge exists. The first edge is selected arbitrarily in both internal and external tiles. If more than one tile is adjacent, the external first edge is chosen arbitrarily from the first tile counterclockwise containing one or more edges.

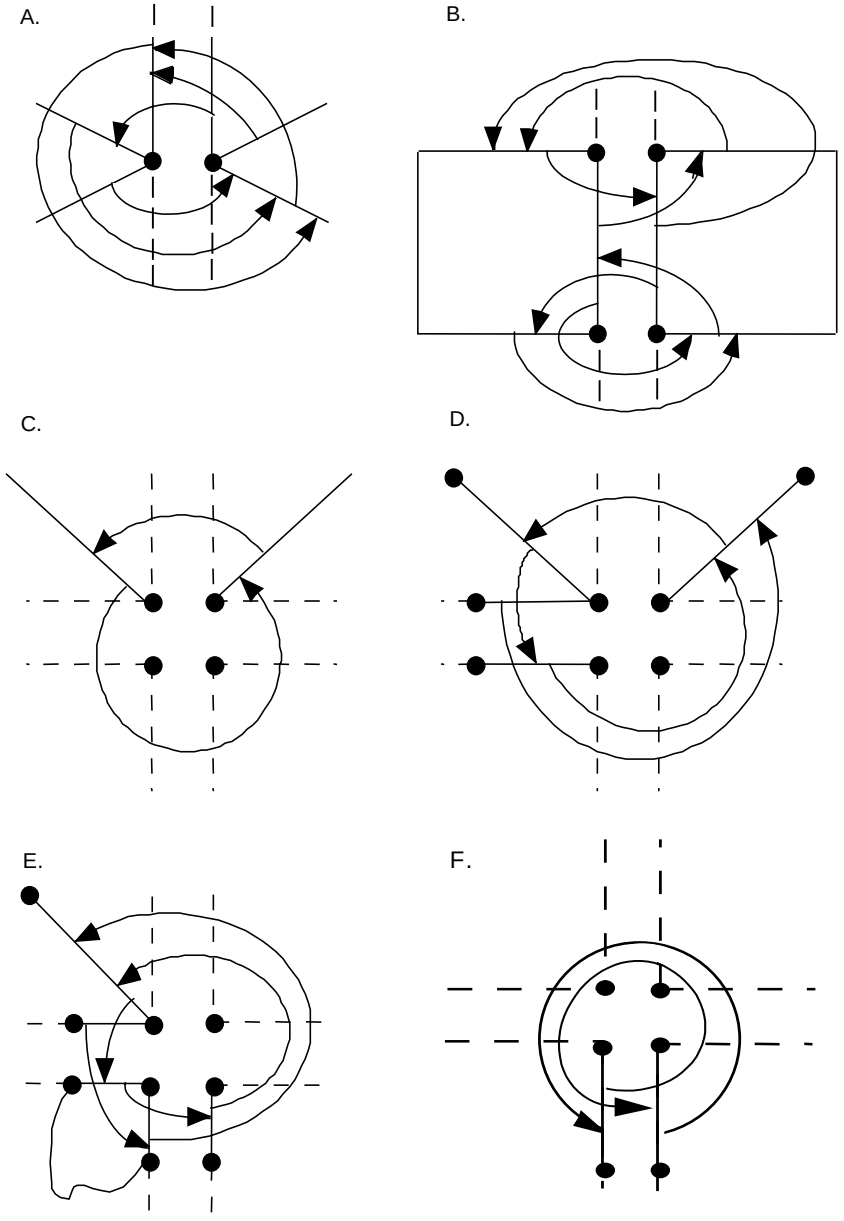


FIGURE 33. Cross-tile edge rules

The following are additional examples of tile boundary primitive behavior:

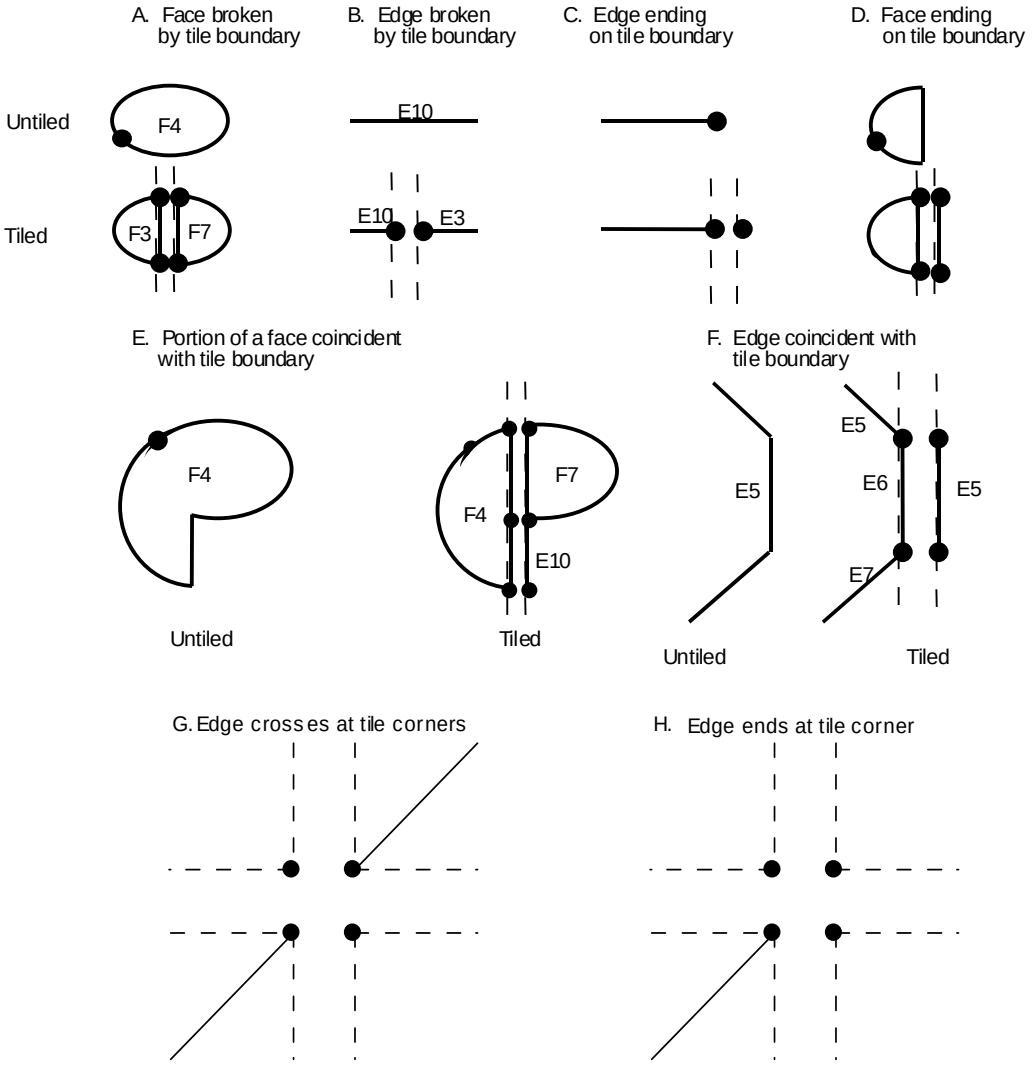
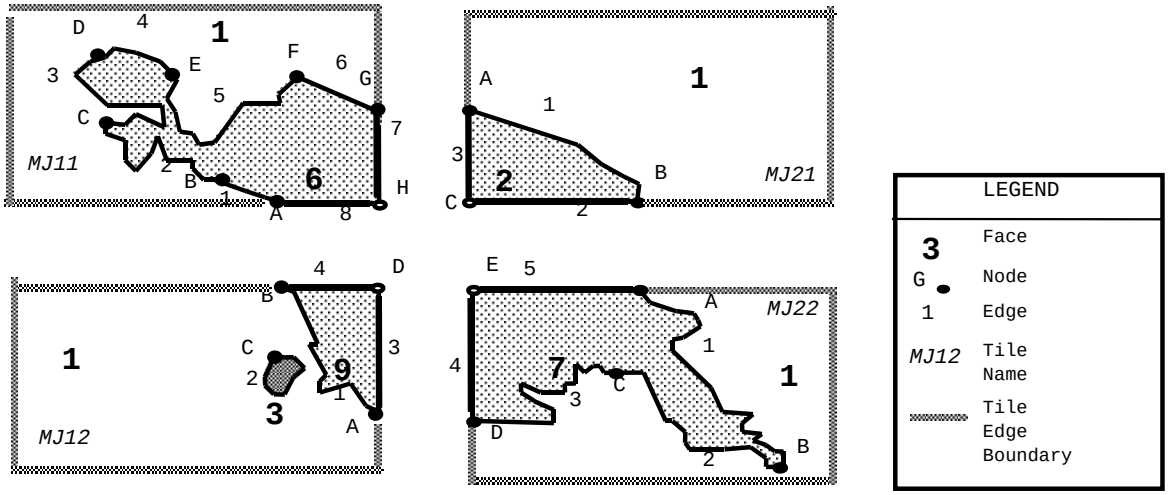


FIGURE 34. Tile boundary primitive behavior.



TILE MJ21

ID	Start Node	End Node	Right Face Face,Tile,ExFace	Left Face Face,Tile,ExFace	Right Edge Edge,Tile,ExEdge	Left Edge Edge,Tile,ExEdge	Coordinates
1	A	B	2,-,-	1,-,-	2,MJ22,5	3,MJ11,6	Not Shown
2	B	C	2,-,-	1,MJ22,7	3,MJ11,7	1,MJ22,1	
3	C	A	2,-,-	1,MJ11,6	1,MJ11,6	2,MJ11,8	

TILE MJ22

ID	Start Node	End Node	Right Face Face,Tile,ExFace	Left Face Face,Tile,ExFace	Right Edge Edge,Tile,ExEdge	Left Edge Edge,Tile,ExEdge	Coordinates
1	A	B	7,-,-	1,-,-	2,-,-	5,MJ21,1	Not Shown
2	B	C	7,-,-	1,-,-	3,-,-	1,-,-	
3	D	C	1,-,-	7,-,-	2,-,-	4,MJ12,3	
4	D	E	7,-,-	1,MJ12,9	5,MJ21,2	3,MJ12,1	
5	E	A	7,-,-	1,MJ21,2	1,MJ21,1	4,MJ21,3	

NOTE: Tile names are shown for clarity. The triplet id actually contains the tile id.

FIGURE 35. Cross-tile edge example.

a. Start with tile MJ21, edge 1. Read the left edge. Choose to cross into tile MJ11, edge 6.
b. Chain from edge 6, 5, 4, 3, 2, and 1 within tile MJ11.
c. From edge 1 in MJ11, go across to tile MJ12, edge 1.
d. From edge 1 in MJ12, cross tiles into tile MJ22, edge 3.
e. Chain through edges 3, 2, and 1.
f. From edge 1 in MJ22, cross into tile MJ21, edge 1.
g. When the end of the face cycle is reached, exit.

REPRINTED WITHOUT CHANGE

SPATIAL INDEXING

F.1. GENERAL

F.1.1 Scope. This appendix provides information and discussion concerning spatial indexes in a VPF database. The information contained in this standard shall be used by the Military Departments, Office of the Secretary of Defense, Organizations of the Joint Chiefs of Staff and the Defense Agencies of the Department of Defense (collectively known as DoD Components) in preparing and accessing digital geographic data required or specified to be in vector product format.

F.2. APPLICABLE DOCUMENTS

This section is not applicable to this appendix.

F.3. DEFINITIONS

For purposes of this appendix, the definitions of section 3 of the main document shall apply.

F.4. GENERAL INFORMATION

F.4.1 Introduction. Spatial queries are queries in which the user points at a specific position on a display device containing a graphic representation of the data and asks (for example) "What is this line?" In order to answer the spatial query, any software that conducts a spatial query on a VPF database must search the edge primitive table for an exact match with "this line." Without a spatial index, the software would have to search every vertex of every edge sequentially for the correct response.

The purpose of a spatial index is to improve the speed with which software can retrieve a specific set of row ids from a primitive table. If the database contains spatial indexes, the software, when given a spatial query like, "What are the features within this bounding region?" can quickly retrieve the primitives that match the query. For each primitive (face, edge, entity node, connected node, and text), there can exist a spatial index file: fsi, esi, nsi, csi, or tsi (see section 5.4.2).

F.4.2 Categories of spatial decomposition. The spatial index is the second of four categories of spatial decomposition of a VPF database. The other three are the tile directory, the minimum bounding rectangle of the edge and face primitives, and the primitive coordinates. All four categories of spatial decomposition are described below.

F.4.2.1 Tile directory. Tiles in an implementation of VPF maintain spatially distributed primitives in separate directories. Thus, software developed for a tiled VPF database can search for data in only the relevant tile after the appropriate tile has been identified.

F.4.2.2 Spatial index. The second step in a typical software query is to use the appropriate index file (if one has been created within the database design). It is recommended that spatial index files associated with the primitives be created for every product implementation of VPF. Spatial indexes are discussed further below.

F.4.2.3 Minimum bounding rectangle (MBR). VPF requires that face and edge primitives have associated bounding rectangle table files—FBR and EBR. These tables allow the rapid retrieval of the primitives' spatial extent and are used by the software after the spatial index routine generates the primitive ids for the current spatial query. The bounding rectangle coordinates are typically used by the software to check the validity of the primitive ids in satisfying the query.

F.4.2.4 Primitive coordinates. It is necessary for software to exhaustively check nodes and text primitives for satisfaction of a spatial query, since these primitives do not have associated minimum bounding rectangles. The coordinate of the primitive is thus used to ensure the accurate retrieval of primitive ids output from the spatial index.

F.4.3 VPF spatial index file. The spatial index file internal structure in VPF is based on an adaptive grid binary tree. This method is powerful because it can handle all types of spatial queries (point, line, and area). The input primitives are broken down into a grid-based binary tree. At each cell (of the tree) there is a list of primitive MBRs and a list of the primitive ids that are found at this level of the tree.

The tree is created by storing primitive ids at a cell of the tree. "Bucket size" is the number used to determine when to split a cell and is defined by the product specification. A typical bucket size is eight (8). If the cell fills to the bucket size, then the cell is split into right and left (or top and bottom) children of the cell. The primitive ids are then distributed down into either child depending on the primitive MBRs. Only if a primitive MBR intersects the adjoining child’s cell, will the primitive id remain in the parent cell. Since primitives cannot be split between two cells, the number of primitives stored in a cell may exceed the bucket size. The process of splitting cells and distributing primitives based on each primitive's MBR and the

bucket size continues until no cell requires splitting or the subdivision process reaches single dimensioned cells(max and min x,y are equal). See F.4.3.2 below for further detail on the spatial index coordinate system. Product Specifications may limit the number cells in the grid-based binary tree for performance reasons (see F.4.4.1)

When examined spatially, the spatial index divides a tile into sub elements (the cells of the tree); FIGURE 51. Each split results in dividing the parent cell into half. The first split is into right and left halves, with the left half x-axis ranging from 0 to 127 and the right half x-axis ranging from 128 to 255. The next split is into top and bottom halves, with the bottom half y-axis ranging from 0 to 127 and the top half y-axis ranging from 128 to 255. Splits then continue, left/right and top/bottom, until the tree is complete.

The actual format of the spatial index file consists of the following:

a. A header containing the number of primitives, the minimum bounding rectangle of the entire spatial extent of the tree, and the number of cells in the tree. Note that, in tiled coverages, the MBR of the entire spatial extent of the tree will coincide with the tile boundary in topology level three coverages for face spatial indices (fsi), edge spatial indices (esi), connected node spatial indices (csi), and text spatial indices (tsi). In untiled topology level three coverages, the entire spatial extent of the tree will coincide with the coverage extent. However, there is no reason to force the tile boundary as the spatial extent of the grid-based binary tree for entity node spatial indices (nsi) or for coverages with topology levels less than three. In fact, using the primitives MBR as the spatial extent of the tree rather than the tile boundary provides for a more efficient spatial index. Although the example spatial index described in this Appendix is for a tiled coverage, the concepts would apply to an untiled coverage as well.

b. A bin array of the tree. Each bin contains two items. A beginning location (offset from the end of the BIN Array Record) for the cell’s data and the number of primitives in the cell. All intermediate cells are listed, even if empty. The offset for an empty cell equals zero. The last entry in the bin array record is always a populated cell. The final level of the tree is not forced to be balanced by creating empty cells.

c. Data records for each primitive in the tree. There is one record for each primitive in the tree. Each record contains four 1-byte integers defining the MBR for a primitive and that primitive’s id.

F.4.3.1 Tree navigation. For any cell, the cell from which it was generated is the integer value obtained by dividing by two. Thus, cell 3 points back to cell 1 [INT(3/2)], as does cell 2.

New cells created by splitting are numbered by multiplying the current cell by two and adding one for the second new cell. Thus, cell 2 becomes cells 4 and 5, and cell 3 becomes cells 6 and 7.

F.4.3.2 Spatial index coordinate system. The coordinate system for the spatial index is based upon 1-byte integers, so a primitive's MBR must be converted to the spatial index coordinate system. All coordinates are relative to the southwest corner of the tile or of the MBR of the data extent for the primitive type (see F.4.3a), and will range from 0 to 255. The minimum X and Y axis coordinate for each cell will be zero (0) or an even integer. The maximum X and Y axis coordinate for each cell will be an odd integer. The only exception to this rule is at level 16, where x-min = x-max and y-min = y-max. For any level of cell decomposition, therefore, a single integer value will fall in only one cell.

There is no single-line boundary between cells. The number of cells and cell dimensions at each decomposition level are shown below.

Decomposition Level	Bins at Level	Total bins In Index	Subdivision	Cell X,Y Dimension
Level 0	1	1	No partition	256 X 256
Level 1	2	3	Vertically	128 X 256
Level 2	4	7	Horizontally	128 X 128
Level 3	8	15	Vertically	64 X 128
Level 4	16	31	Horizontally	64 X 64
Level 5	32	63	Vertically	32 X 64
Level 6	64	127	Horizontally	32 X 32
Level 7	128	255	Vertically	16 X 32
Level 8	256	511	Horizontally	16 X 16
Level 9	512	1023	Vertically	8 X 16
Level 10	1024	2047	Horizontally	8 X 8
Level 11	2048	4095	Vertically	4 X 8
Level 12	4096	8191	Horizontally	4 X 4
Level 13	8192	16383	Vertically	2 X 4
Level 14	16384	32767	Horizontally	2 X 2
Level 15	32768	65535	Vertically	1 X 2
Level 16	65536	131071	Horizontally	1 X 1*

* This represents a single dimensioned cell with the min x,y equal to the max x,y.

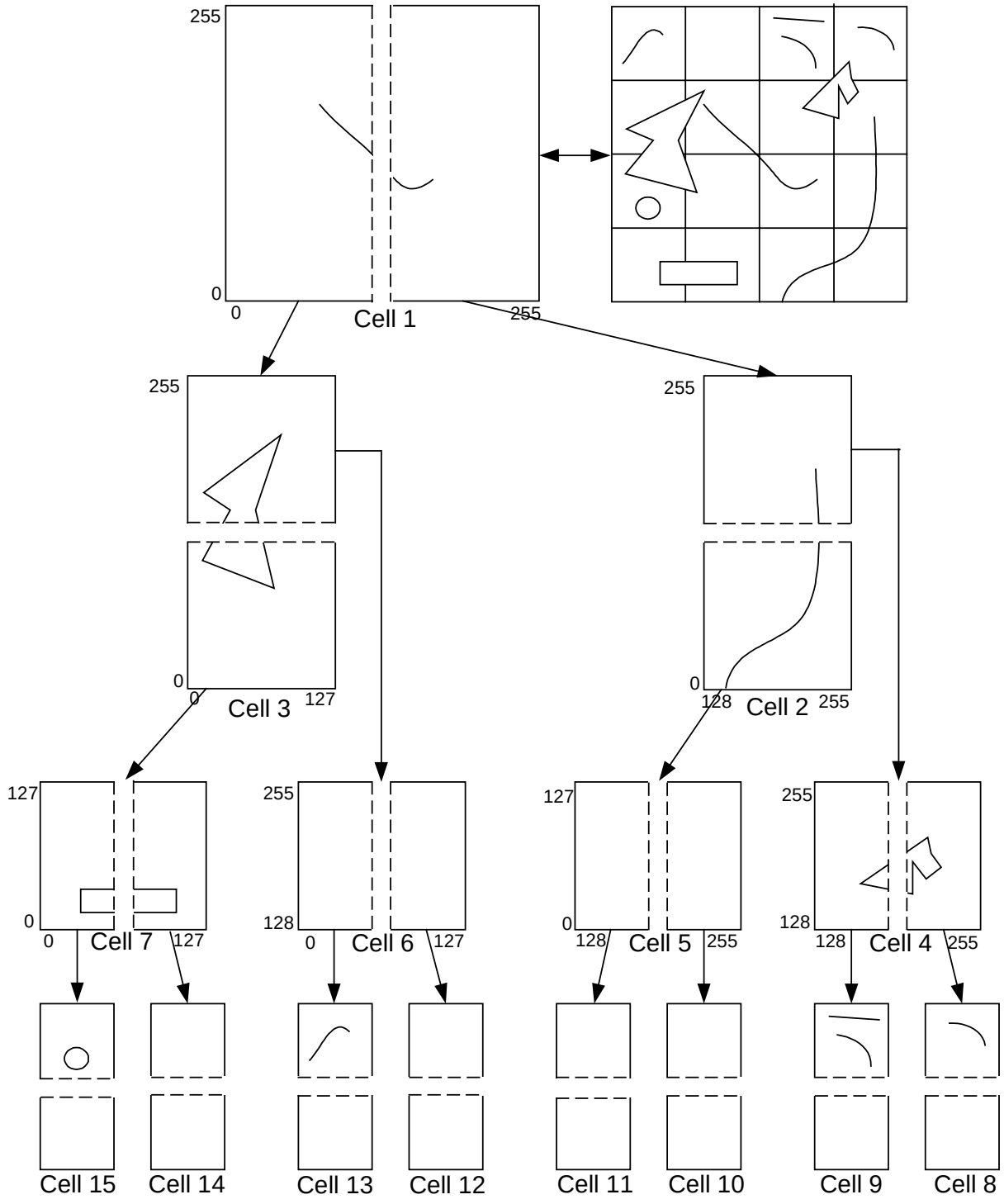


FIGURE 51. Spatial index cell decomposition.

F.4.4 Examples of spatial index creation. TABLE 69 is a listing of the minimum (x_1, y_1) and maximum (x_2, y_2) coordinates of the MBRs of 19 face primitives. The coordinate values in TABLE 66

TABLE 69. Minimum and maximum coordinates for 19 primitives in a tile. Universe is primitive number 1.

Primitive ids	x_1 (deg)	y_1 (deg)	x_2 (deg)	y_2 (deg)
1	Null	Null	Null	Null
2	-5.00	54.63	-3.57	55.00
3	-5.00	52.00	-2.74	55.00
4	-5.00	54.91	-4.99	54.94
5	-5.00	54.76	-4.99	54.77
6	-4.80	54.06	-4.31	54.42
7	-3.28	54.05	-3.17	54.15
8	-0.71	53.54	0	53.74
9	-0.57	53.68	-0.53	53.69
10	-4.60	53.13	-4.05	53.43
11	-4.71	53.24	-4.56	53.33
12	-4.80	52.75	-4.78	52.77
13	-5.00	50.53	-2.35	51.82
14	-4.71	51.63	-4.68	51.65
15	-4.68	51.16	-4.65	51.20
16	-1.03	50.78	-0.95	50.84
17	-1.59	50.58	-1.08	50.77
18	-1.99	50.69	-1.96	50.70
19	-5.00	50.16	-4.99	50.17

are all in degrees. The primitives are all located within a 5- by 5-degree tile that has an MBR of (-5, 50), (0, 55). The MBR coordinates can be converted to the spatial index coordinate system as follows:

For minimum (x_1,y_1) and maximum (x_2,y_2) each new coordinate is obtained from the integer truncation of

255*(Original coordinate – minimum)/(maximum – minimum), which much be in the range 0 to 255.

In our example with mbr (-5 , 50), (0 , 55), each new coordinate is given by

y = 255*(Latitude – 50)/(55 – 50) truncated to an integer

and

x = 255*(Longitude + 5)/(0 + 5) truncated to an integer.

The results of this conversion are listed in TABLE 70.

TABLE 70. Minimum and maximum spatial index coordinates.

Primitive ids	x_1	y_1	x_2	y_2
2	0	236	72	255
3	0	102	115	255
4	0	250	0	251
5	0	242	0	243
6	10	207	35	225
7	87	206	93	211
8	218	180	255	190
9	225	187	227	188
10	20	159	48	174
11	14	165	22	169
12	9	140	11	141
13	0	27	135	92
14	14	83	16	84
15	16	59	17	61
16	202	39	206	42
17	173	29	199	39
18	153	35	155	35
19	0	8	0	8

NOTE: Since face 1 (universe face) is a topological artifact (i.e. no outer ring), the MBR in normalized coordinates cannot be calculated. Therefore face 1 is not included in the bin data record portion of the index. Further, no fsi is built for a face table containing only the universe face.

When coordinates are stored as short floating point (data types C and Z), different computer systems can generate slightly different values due to conversion to long floating point for normalization computations. When this occurs very close to a cell break, it can cause the primitive MBR normalized coordinates to fall in an incorrect cell. To alleviate this problem, the following process should be followed for computing normalized coordinate for any short floating point coordinate value: after conversion to a long floating point, truncate the coordinate value following the third (3^{rd}) decimal place before computing the normalized coordinate.

If the MBRs of each primitive are plotted, they appear as shown in FIGURE 52. In FIGURE 53, dividing lines have been added to FIGURE 52 to show that primitive 13 is present in both the left and right halves of the tile, and that primitive 3 is present in both the top and bottom halves of the tile.

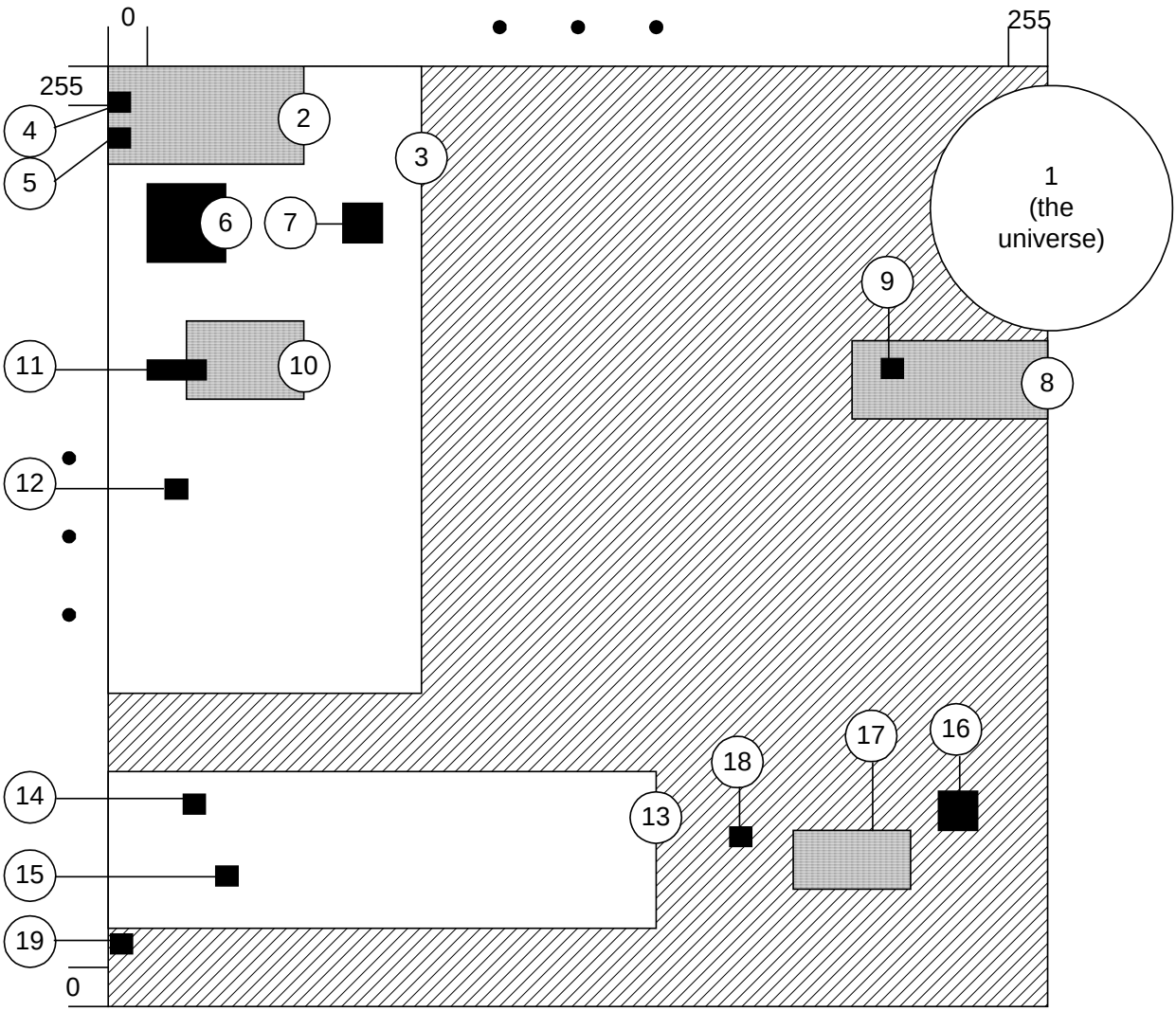


FIGURE 52. Location of MBRs in tile.

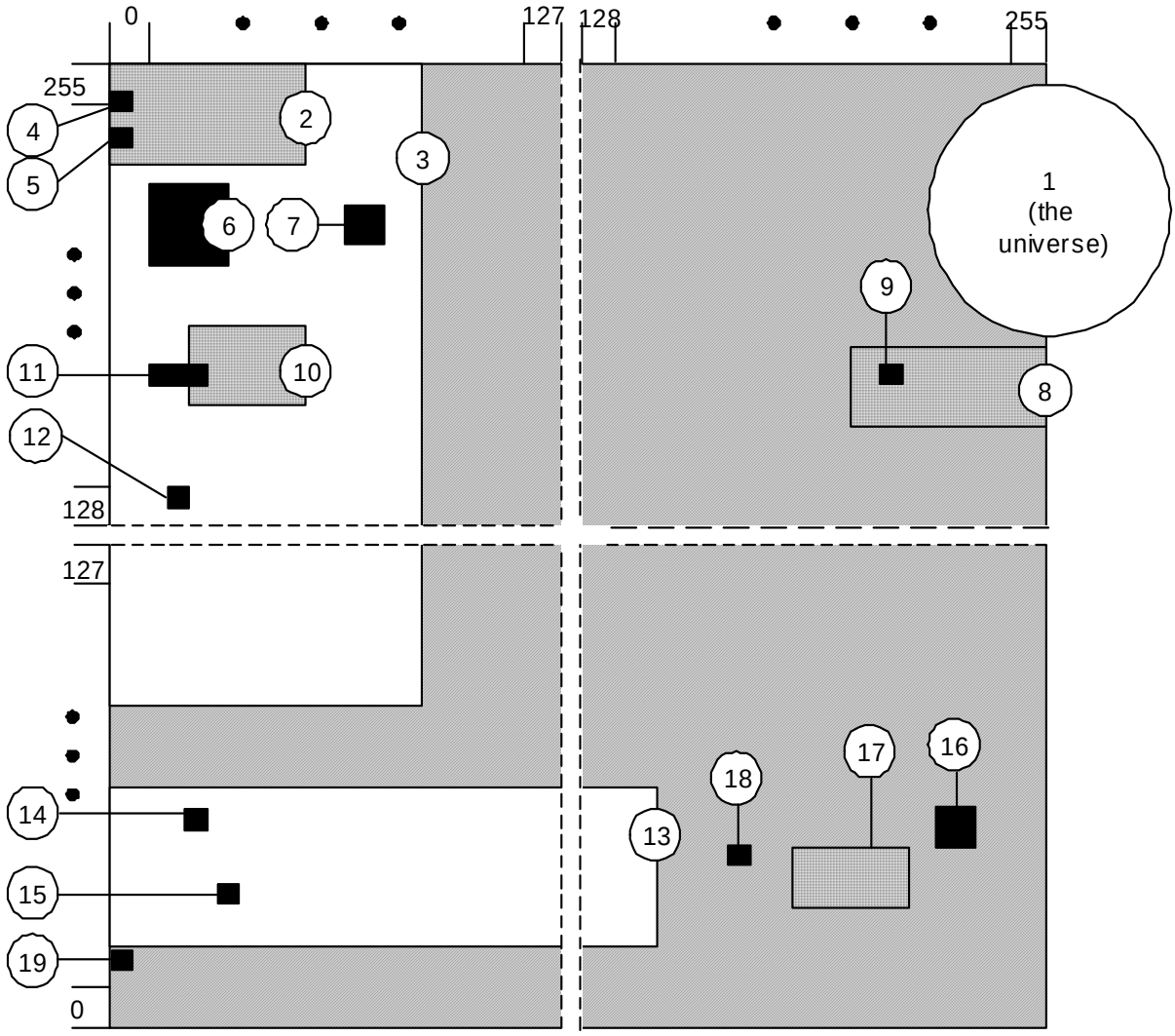


FIGURE 53. Tile content divided in four quarters.

F.4.4.1 Example of tree creation. The bucket size for this example is set at eight. If a parent cell contains nine or more primitives that can be propagated to the next level, the parent splits into two children. The first split of the tile puts primitives 8, 9, 16, 17, and 18 into cell 2 and places primitive 3 into cell 3 (FIGURE 54). Primitive 13 must be held in cell 1 (FIGURE 55) because neither of the split cells contains the entire MBR for primitive 13.

Cell 3 (before the split into cells 6 and 7) contains twelve primitives: 2, 3, 4, 5, 6, 7, 10, 11, 12, 14, 15, and 19. Since this exceeds the defined bucket size, cell 3 is split (FIGURE 54). The MBR of primitive 3 will cross the boundary of cells 6 and 7.

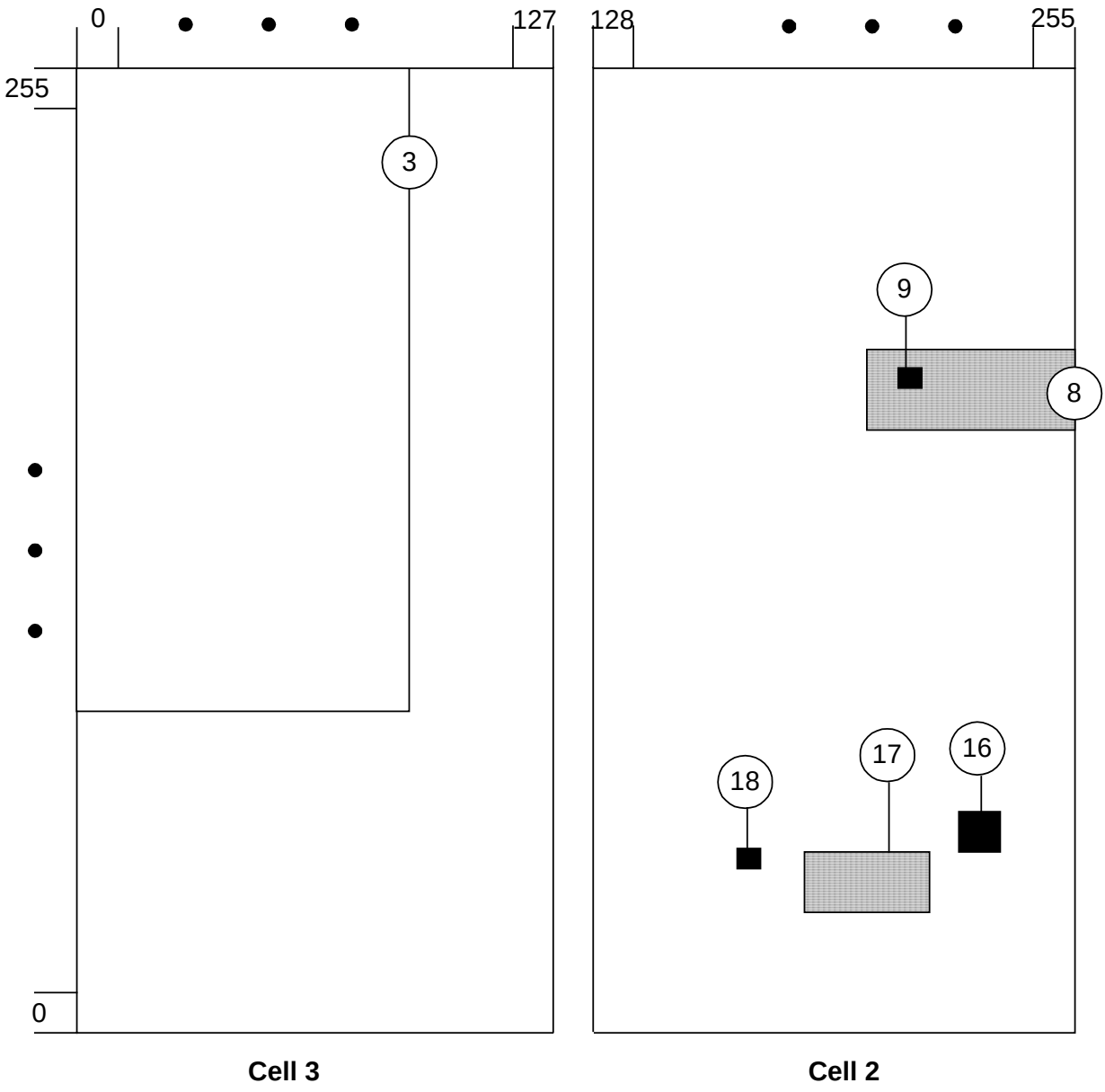


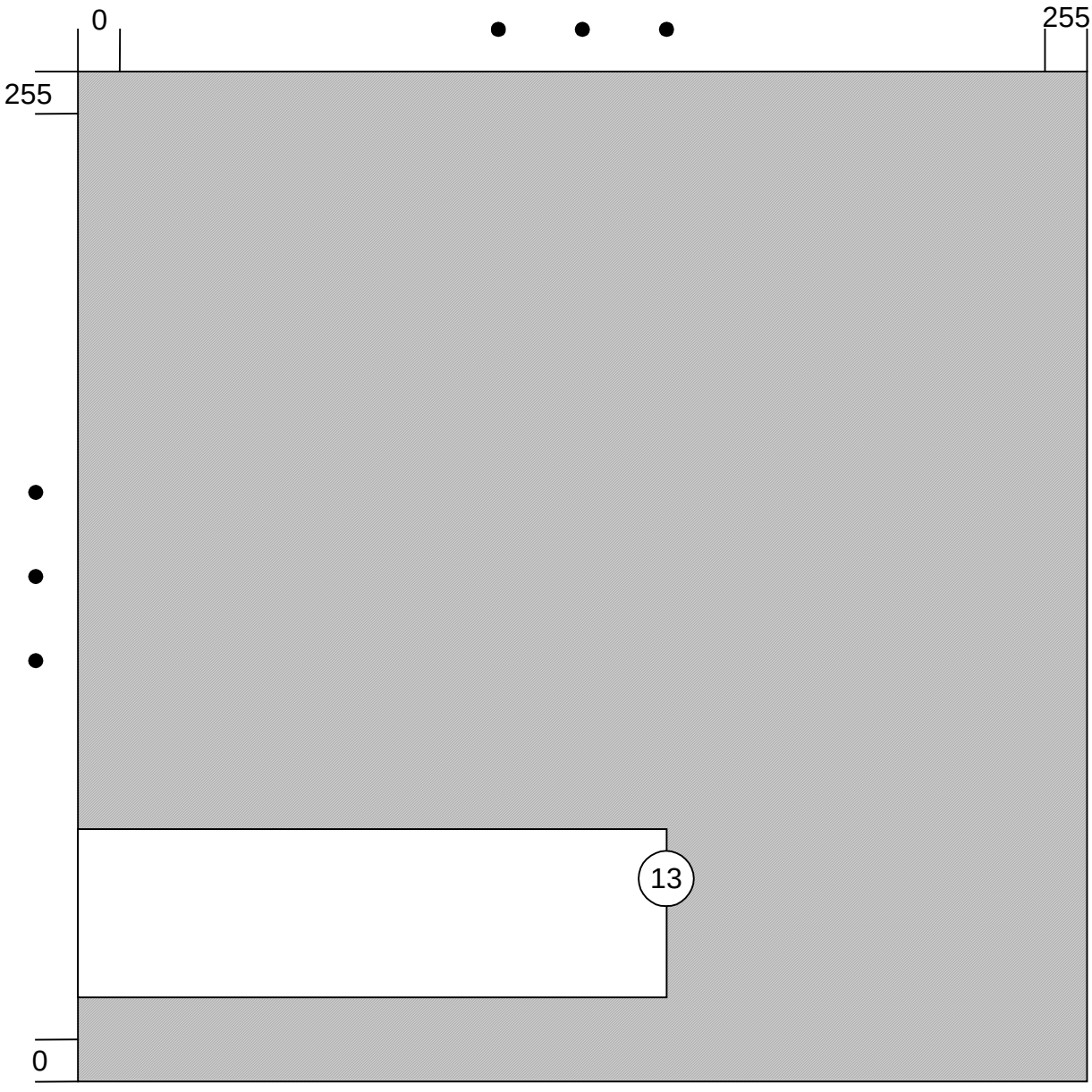
FIGURE 54. First cell split.

Therefore it must remain in cell 3. Eleven primitives remain. Based on each primitive’s MBR, the contents of the new cells are: primitives 2, 4, 5, 6, 7, 10, 11, and 12 in cell 6 and primitives 14, 15, and 19 in cell 7. Note that no primitives are allocated to cells 4 and 5, since all five primitives on the right half of the tile could be held in cell 2. The five primitives do not ’over-flow’ the defined bucket size. All of the primitives in this example have now been allocated to the tree.

In theory, the process of splitting cells and distributing primitives based on each primitive’s MBR and the bucket size

continues until no cell requires splitting or the subdivision process reaches the bottom level of the grid-based binary tree (where each cell is 1x1 in normalized coordinates). This implies a grid-based binary tree with 17 levels and 131,071 cells (2^{17} - 1). As noted in E.4.3, however, Product Specifications may limit the number of cells allowed in the grid-based binary tree for performance reasons.

The spatial index that results for this example is shown in TABLE 71.



Cell 1

FIGURE 55. Content of cell 1.

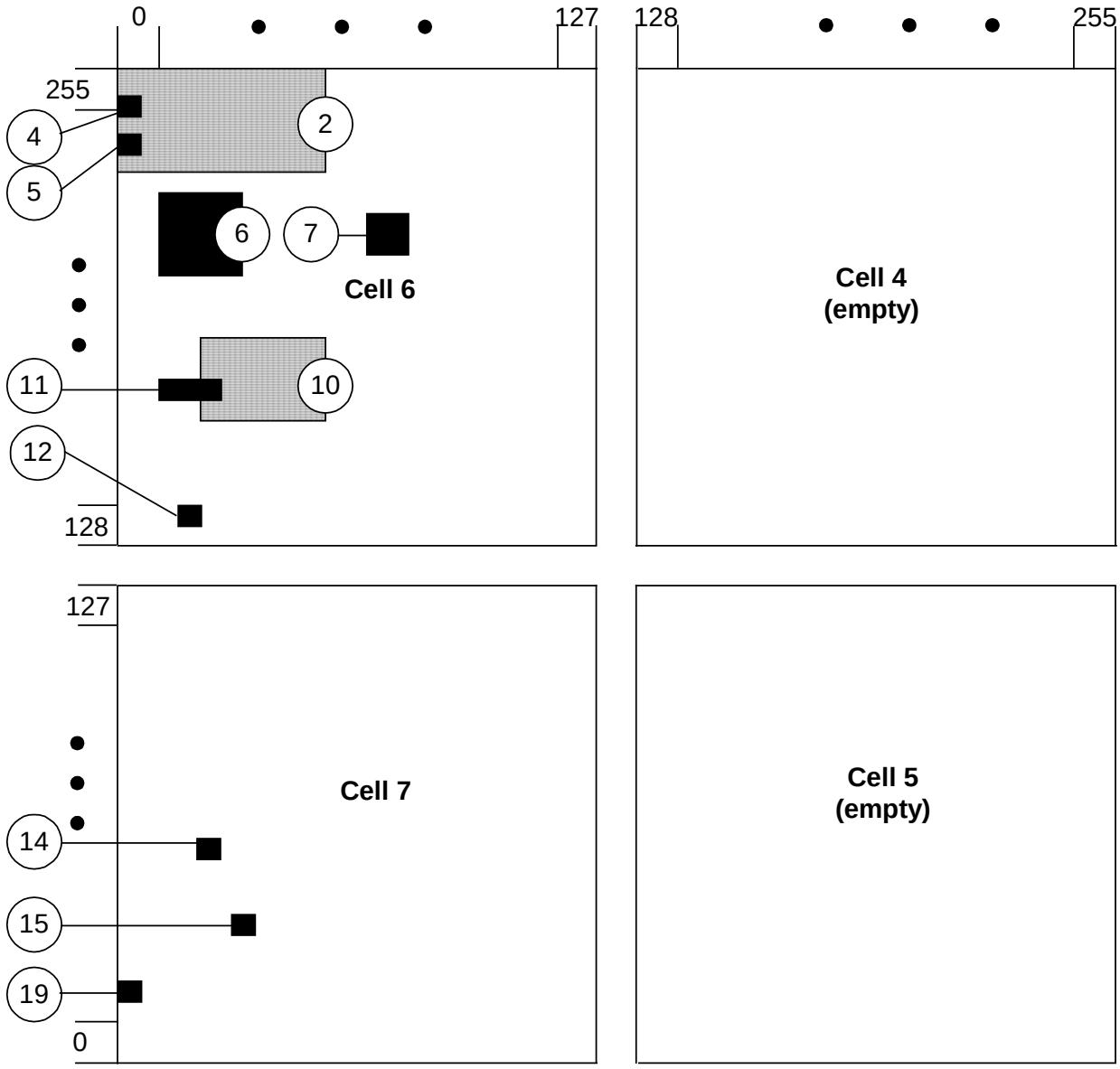


FIGURE 56. Second cell split.

TABLE 71. Example of spatial index.

Header:

Number of primitives: 18
MBR: -5.00, 50.00, 0.00, 55.00
Number of cells: 7

Bin Array Record:

Cell (information not in actual spatial index)	Offset	Primitive count
1	0	1
2	8	5
3	48	1
4	0	0
5	0	0
6	56	8
7	120	3

Bin Data Record:

Record Number	Offset Address	x_1	y_1	x_2	y_2	Primitive ids
Cell 1						
1	0	0	26	135	93	13
Cell 2						
2	8	153	35	155	35	18
3	16	173	29	199	39	17
4	24	202	39	206	42	16
5	32	226	187	227	188	9
6	40	218	180	255	190	8
Cell 3						
7	48	0	102	115	255	3
Cell 6						
8	56	87	206	93	211	7
9	64	10	206	35	225	6
10	72	0	242	0	243	5
11	80	0	250	0	252	4
12	88	0	236	72	255	2
13	96	20	159	48	175	10
14	104	14	165	22	169	11
15	112	9	140	11	141	12
Cell 7						
16	120	0	8	0	8	19
17	128	16	59	17	61	15
18	136	14	83	16	84	14

F.4.5 Spatial query. A spatial index can support a spatial query in several ways. For node primitives, software may require the point to be within a specified distance of a pixel designated

during the query process, or within a box generated during the query process. For an edge primitive, the software may specify that candidate edges are to be within some specified perpendicular distance of the query pixel or have MBRs that intersect a query box. For a face primitive, the software may define the candidate edges as having MBRs that intersect a query box, be fully contained within the query MBR, or be determined by solving the "point-in-polygon" puzzle. In any case, the spatial index forms the starting point for the database search. It works as follows:

a. The user designates a query point (pixel).

b. The software converts the query point into the spatial index coordinate system ((0,0) to (255,255)).

c. The software tests all features within the top level cell, if any, to determine if these features qualify for the query response.

d. The software calculates the next smaller cell containing the query point.

e. The software repeats the test (if necessary) and continues to decrease cell size until no more records exist.

F.4.6 Spatial query using the sample tree.

a. The user designates a query point and the software determines the normalized coordinates at (192, 32).

b. Beginning at the root of the tree, the software goes to cell 1 and finds face 13.

c. The software checks the MBR of face 13 to determine if it includes the query point.

d. Since face 13 does not include the query point the software calculates the children of cell 1, which are cells 2 and 3. Cell 3 cannot contain the query point, so cell 2 is examined. Faces 8, 9, 16, 17, and 18 are found.

e. The software checks the MBR of these faces to determine if they include the query point.

f. The software reports the query result which is face 17.